# Data Toolkit

V4 Documentation

# Table of Contents

# Overview

Ever needed the ability to create a data file on-the-fly with Switch metadata? Join multiple files together into a single JSON or XML file? Convert an XML file into a CSV or Excel file? Or, perhaps, you're familiar with other similar automation software like Switch that has similar functionality. This is the answer.

**Data Toolkit** is a collection of apps (or tools) to select. create, manipulate, and convert documents for ease of use within Switch Flows. They were also designed to work together to provide more benefit than just utilizing them separately.

If there is functionality you believe is missing or have an idea on how to expand the current functionality, please reach out to us by sending us an email with the subject prefix "*Data Toolkit Functionality*".

**Simple Use-Cases**

- Create a JSON/JDF/XML file to send for output to another application or RIP.
- Edit existing JSON/XML by adding new information to the file before saving out to a job folder.
- Split on multi-item jobs to send through Switch individually.

**Multi-App Use-Cases**

- Create a CSV of batched incoming jobs by using the **Create Data** app to create the individual XML's, **Join Data** app to join these files together, and **Convert From XML** app to make a CSV document.

# About Significans Automation

Significans Automation is a software integrator specializing in delivering next-generation automation to the Printing and Packaging industry.

We offer programming and expertise in custom workflow development, deployment of communication and project management systems, color management, and end-to-end business integration. While upholding software neutrality, Significans Automation advises and tailors best in class software to optimally fit the environment.

The level of sophistication that is provided increases profitability, improved quality control, and enhanced production efficiency, enabling Artificial Intelligence and Robotics, while also facilitating new revenue opportunities in e-commerce. We are driven by the conviction that customized automation is the only path forward.

# Global Requirements & Shared Properties

The following properties and connection behaviors are shared across multiple Apps in this toolkit. They function identically across all applicable Apps unless otherwise noted.

## Compatibility

Switch 25.11 and higher. Windows and macOS.

## Incoming Connections

If no mention of Incoming Connection requirements is specified per app, assume that at least one incoming connection is required.

## Outgoing Connections

All apps require at least one outgoing connection.

## Shared Flow Element Properties

| Property | Type | Description | Example / Usage |
|---|---|---|---|
| **Action on** | Enum | Determines if the App processes the incoming Job or a specific Dataset attached to the job. | Job, Dataset |
| **Dataset** | String | The name of the dataset to process (if Action on is set to Dataset). | LogData |
| **Variables** | String | Multi-line text for variable substitution. Format: key=value. | OrderNo=[Job.PrivateData:Key="Order"] |
| **Scriptable variables** | Boolean | Enables 10 additional dynamic script expression fields for variable creation. | Yes, No |
| **Incoming type** | Enum | Specifies the format of the incoming file for automatic conversion to queryable XML. | XML, JSON, PDF, Spreadsheet, Text |
| **Encoding** | Enum | Defines the character encoding of the incoming/outgoing file. | utf8, ascii, latin1, utf16le |
| **Namespaces** | String | Key-value pairs for XML namespaces (prefix=uri). Use Automatic to let the App infer them. | Automatic or soapenv=http://... |
| **Encode variables** | Boolean | If Yes, variables injected into XML are automatically encoded to be XML compliant. | Yes |

## Spreadsheet Specific Properties

When **Incoming type** is `Spreadsheet`, the following properties become available:

| Property | Description | Example / Usage |
|---|---|---|
| **Sheet by** | Target the sheet by Name or Index. | Name |
| **Index / Sheet** | The numerical index (starting at 1) or exact name of the sheet. | 1 or Sheet1 |
| **Password** | Password to decrypt the spreadsheet (if required). | P@ssw0rd |
| **Range** | Specify ranges as key-value pairs (e.g., customers=A1:D10). Automatic parses the whole sheet. | order=Z1 |
| **Maintain formatting** | If No, raw data is extracted. If Yes, visual cell formatting is maintained. | No |
| **Trim whitespace** | Removes leading/trailing spaces from extracted cell data. | Yes |

## Shared Outgoing Connection Properties

| Property | Description | Example / Usage |
|---|---|---|
| **Move on** | Trigger the connection on Success or Error. | Success |
| **Output** | Choose whether to send the Outgoing (processed) file or the Incoming (original) file. | Outgoing |
| **Type** | Defines if the output job is a Child of the incoming job, or a New Job (no inherited metadata). | Child |
| **Handle datasets** | Dictates how existing datasets are managed. | Keep All, Remove All, Keep Some |
| **Output name** | Defines the resulting filename. Options include Job, Connection, XPath, or Custom. | Job |
| **Attach as dataset** | Attaches the resulting data as a dataset to the outgoing job. | Yes |

## Template Engine

This reference defines the functions available for dynamic data within the templating engine. These functions allow for the cleaning, formatting, and transformation of XML node names, values, and attributes. See appendix for available functions.

| Target | Syntax | Description |
|---|---|---|
| **Current String Value** | s.func() | Targets the current string value. *Only available in Select Data.* |
| **Variable** | v[name].func() | Targets the variable string value. |
| **Node Name** | n.n.func() | Targets the tag name of the element. |
| **Node Value** | n.v.func() | Targets the text content inside the element. |
| **Node Attribute** | n.a[name].func() | Targets the value of a specific attribute. |
| **Node Index** | n.i | Targets the current 1-based loop index. |

# Select Data

## Description

Select key pieces of information within an incoming document (JSON / PDF / Spreadsheet / Text / XML) as private data. Enabling XPath 2.0 functionality, you can go beyond the base Switch metadata functionality and use all available XPath 2.0 functions!

## Flow Element Properties

| Property | Description | Example |
|---|---|---|
| **Handler** | Defines how to handle an XPath query that returns multiple nodes. | Keep First, Keep Last, Separator |
| **Separator** | Character used to concatenate multiple results if Separator is selected. | \| |
| **Select [1-10]** | The Private Data key name to save the result under. | Order_ID |
| **XPath [1-10]** | The XPath query used to locate the data for this selection. | /Order/ID/text() |
| **Function [1-10]** | Optional string manipulation functions applied to the result (e.g., s.trim()). | s.stripHtml().clean() |

---

# Create Data

## Description

**Create Data** with basic syntax validation and variable encoding for correct XML output (if applicable). Switch metadata fields can be used to create complex structures, including the use of script expressions (with the Scripting Module). This app can also allow you to create simple (non-XML based - e.g. JSON, CSV) text documents for other purposes.

## Incoming Connections

- If Flow Element property "*Triggered*" is set to "Job Arrived". At least one incoming connection required.
- If Flow Element Property "Triggered" is set to "Timer". No incoming connection is required.

## Flow Element Properties

| Property | Description | Example |
|---|---|---|
| **Trigger** | Sets whether the App runs when a Job Arrived or on a Timer. | Job Arrived |
| **Value (Timer)** | Numerical duration for the timer. | 30 |
| **Unit (Timer)** | Time unit for the timer value. | Seconds, Minutes, Hours, Days |
| **Structure** | The raw data framework or template to be created. | <A><B>{{OrderNo}}</B></A> |
| **Type** | Defines the outgoing document format. | XML, JSON, Other |
| **Validate** | Performs a syntax check on the generated structure before outputting. | Yes |
| **Pretty print** | Formats the resulting code with appropriate tabs and newlines. | Yes |

# Edit Data

## Description

**Edit Data** uses a find-and-replace method with XPath to manipulate document structures. Beyond standard replacement, this version includes a powerful **Templating Engine** that allows for dynamic data cleaning and transformation using a specific {{ n.x }} syntax. It also supports **Wildcard** operations for batch-modifying attributes.

## Flow Element Properties

**Note**: Includes Shared Properties for incoming data.

| Property | Description | Example |
|---|---|---|
| **Use function namespace** | Removes the need for prefixing fn: on standard XPath 2.0 functions. | Yes |
| **Outgoing type** | Format of the output file. | XML, JSON |
| **Element names** | Standardizes element casing (e.g., lowercase, Titlecase). | Current |
| **Attribute names** | Standardizes attribute casing. | Current |
| **Keep system attributes** | If No, strips Switch-specific _for or _type attributes. | Yes |

## Action Configurations (Actions 1 through 10)

Users can define up to 10 sequential actions. Each action provides contextual properties based on the selected Action Type:

| Action | Key Sub-Properties | Description |
|---|---|---|
| **Select** | XPath | Selects nodes to be used as variables in later Insert actions. |
| **Insert** | XPath, Type, Name, Location, Value | Inserts a new Element, Attribute, CDATA, etc., at the specified XPath. |
| **Split** | XPath, Find, Node name, Copy attributes | Splits text within a node based on a string/regex and creates new child nodes for each split value. |
| **Join** | XPath, Separator, Location | Gathers text from multiple nodes, joins them with a separator, and places the result in a target location. |
| **Update** | XPath, Type, Name, Value | Overwrites the name or value of an existing Attribute or Element. |
| **Delete** | XPath | Removes the targeted node entirely. |

# Split Data

## Description

**Split** documents into multiple by a common element. Useful for cases where a singular file contains more than one 'thing', you need to act on. For example, a web-2-print solution sends a single xml file with multiple ordered items. Splitting on this document allows you to send multiple out.

## Flow Element Properties

Note: Includes Shared Properties for incoming data.

| Property | Description | Example |
|---|---|---|
| **XPath** | The target node that determines where the split occurs. | /Order/Items/Item |
| **Error on empty results** | Triggers a job failure if the XPath finds 0 matches. | Yes |
| **Keep context** | If Yes, retains the ancestor document structure around the split item. | No |
| **Concurrency** | Number of concurrent processes to use when splitting massive files. | Default (1) |
| **Add custom declaration** | Prepend an XML declaration to each newly split file. | No |

## Outgoing Connection Properties

In addition to the standard Output Properties, Split Data includes advanced routing and folder packaging options on its connections:

| Property | Description | Example |
|---|---|---|
| **Save In Folder?** | Packages the resulting split files into an outgoing folder. | Yes |
| **Folder name** | Determines the name of the folder if Save In Folder is Yes. | Job, Custom |
| **Chunk** | The maximum number of files to place in a single output folder before creating a new one. | 100 |
| **Index file(s)** | Appends a sequential index to the output filenames. | Yes |
| **Pad index?** | Pads the index with a string for clean alphabetical sorting. | Yes (Length: 3, String: 0) |
| **Save private data** | Injects split metrics (Count, JobName, XPath) into the job's Private Data. | Yes |

# Join Data

## Description

**Join** documents together from an XPath expression. Merging these documents eliminates the need to keep track of separate documents all together, or to create complex XML structures with ease.

## Flow Element Properties

**Note**: Includes Shared Properties for incoming data.

| Property | Description | Example |
|---|---|---|
| **XPath** | XPath query executed against every file in the incoming folder to extract the nodes to merge. | /* |
| **Add custom declaration** | Prepends an XML declaration to the joined document. | Yes |
| **Version / Encoding** | Defines the XML version and encoding for the declaration. | 1.0, UTF-8 |
| **Root element name** | The XML wrapper element that will house the joined nodes. | Root |
| **JSON type** | If outputting JSON, defines if the root should be an Array or an Object. | Array |

---

# Convert From XML

## Description

**Convert From XML** allows you to convert your incoming document to one of several formats. Below is a list of what is available:

## Flow Element Properties

**Note**: Includes Shared Properties for incoming data.

| Property | Description | Example / Usage |
|---|---|---|
| **Outgoing type** | The format the XML file will be converted into. | JSON, Spreadsheet |
| **XPath** | (Spreadsheet only) Targets the specific XML node level to convert into rows/columns. | /Root/Order/Item |
| **Use function namespace** | Removes the need for prefixing fn: on standard XPath 2.0 functions. | Yes |
| **Keep incoming attributes** | (JSON only) Retains XML attributes in the resulting JSON object. | Yes |
| **Strip prefix** | (JSON only) Removes namespace prefixes from node names during conversion. | No |
| **Trim whitespace** | Removes leading/trailing spaces from the outgoing data values. | Yes |
| **Custom headers** | (Spreadsheet only) Maps XML node names to custom column headers. Format: xmlNodeName=New Column Name. | itemId=Product ID |
| **Type** | (Spreadsheet only) Specific file format to output. | Excel 2007+ XML Format, Comma Separated Values |

| | | |
|---|---|---|
| **Sheet name** | (Spreadsheet only) Name of the worksheet tab (for supported formats). | Sheet 1 |
| **Compression** | (Spreadsheet only) Enables file compression for supported formats (e.g., .xlsx). | No |
| **Properties** | (Spreadsheet only) Unlocks standard document metadata fields. | Yes |
| **Custom properties** | (Spreadsheet only) Adds custom metadata to the spreadsheet file. Format: key=value. | Department=Prepress |
| **Delimiter** | (CSV/TXT only) Character used to separate column values. | Comma (,), Tab (\t) |
| **Separator** | (CSV/TXT only) Line ending style. | Carriage Return with Line Feed (\r\n) |
| **Pretty print** | (JSON only) Formats the resulting JSON with tabs and newlines for readability. | No |

---

# Convert To XML

## Description

**Convert to XML** from incoming JSON or Excel Spreadsheet document. Outputting a proper XML document that can be attached as a dataset to be utilized in Switch.

## Flow Element Properties

**Note**: Includes Shared Properties for incoming data.

| Property | Description | Example / Usage |
|---|---|---|
| **Incoming type** | Specifies the format of the incoming file to parse. | JSON, PDF, Spreadsheet, Text |
| **Keep system attributes** | Retains structural attributes generated during conversion (e.g., _type="array"). | Yes |
| **Trim whitespace** | Removes leading and trailing spaces from all generated XML text nodes. | No |
| **Add custom declaration** | Prepends an XML declaration to the top of the output file. | Yes |
| **Version** | The XML standard version. | 1.0 |
| **Encoding** | The character encoding declared in the XML header. | UTF-8 |
| **Standalone** | Indicates if the document relies on an external DTD. | Yes |
| **Pretty print** | Formats the resulting XML with appropriate tabs and newlines. | Yes |

# Appendix

## A: Template Engine Examples

| Scenario | Input | Template | Resulting Output |
|---|---|---|---|
| **Remove System Prefix** | N/A | {{ n.n.removePrefix("custcol_di_") }} | Removes specific prefix from node name |
| **Math-Ready Value** | N/A | {{ n.v.mathNumber() }} | Value with only numbers and decimals |
| **Safe Attribute Name** | N/A | {{ n.a[sku].clean().limit("10") }} | Cleaned and truncated attribute value |
| **Clean messy HTML name** |  \t Smith & Sons™ <br/> 123 Main St.  | {{ n.v.stripHtml().decodeHtml().asciiOnly().clean() }} | Smith & Sons 123 Main St. |
| **Conditional Date Stamp** | Rush Order | {{ n.v.default("Standard Processing").now("append", " - ") }} | Rush Order - 2024-10-24 |
| **Defensive XML Naming** | 1st Item (Urgent!) | {{ n.n.textOnly().snakeCase().invalidName("item_node") }} | st_item_urgent |

## B: Variable String Functions

When utilizing variables or modifying data in properties that support functions (like the Function property in the Select Data App, or variable injection brackets {{...}}), you can append the following commands to manipulate strings dynamically.

| Function | Description | Example / Result |
|---|---|---|
| **asciiOnly()** | Strips all non-ASCII (Unicode) characters. | Café → Caf |
| **camelcase()** | Converts a string to camelCase. | Pre Press → prePress |
| **capitalize()** | Capitalizes the first letter of each word. | rush order → Rush Order |
| **clean()** | Collapses multiple spaces, tabs, and newlines into a single space. | Hello \n World → Hello World |
| **decodeHtml()** | Replaces HTML entities with standard characters. | <Tag> → <Tag> |
| **default("val")** | Applies a fallback value if the string is entirely empty. | "" → val |
| **euroNumber()** | Converts EU number formatting to standard float. | 1.500,50 → 1500.50 |
| **extractLetters()** | Removes all numbers and symbols. | 123 Main St! → Main St |
| **extractNumbers()** | Removes all letters and non-math symbols. | Qty: 1,500 ea → 1,500 |
| **filenameSafe()** | Replaces illegal OS characters with underscores. | File:Name → File_Name |
| **kebabcase()** | Converts a string to kebab-case. | Pre Press → pre-press |
| **limit(num, "...")** | Truncates a string to a specific length, optionally appending a suffix. | limit(5, "...") on Priority → Pr... |
| **lower()** | Converts the string to all lowercase. | URGENT → urgent |
| **mask(num, "*")** | Masks characters, leaving only the last num characters visible. | mask(4, "*") on 123456 → 3456 |
| **mathNumber()** | Extracts a pure math float (strips commas, keeps decimals). | 1,500.50 → 1500.50 |
| **now("mode")** | Injects the current date (YYYY-MM-DD). Modes: append, prepend, override. | now("append") → File 2026-03-27 |
| **padend(num, "char")** | Pads the end of the string to reach the target length. | padend(5, "0") on 12 → 12000 |
| **padstart(num, "char")** | Pads the start of the string to reach the target length. | padstart(5, "0") on 12 → 00012 |
| **pascalcase()** | Converts a string to PascalCase. | pre press → PrePress |
| **removePrefix("x")** | Removes a specific string only if it exists at the start. | removePrefix("ID-") on ID-45 → 45 |
| **replace("x","y")** | Replaces occurrences of string x with string y. Supports Regex. | replace("a","e") on cat → cet |
| **slice(start, end)** | Extracts a section of a string by index. | slice(0, 3) on Switch → Swi |
| **snakecase()** | Converts a string to snake_case. | Pre Press → pre_press |
| **split("x", index)** | Splits a string by delimiter x and returns the item at index. | split("-", 1) on A-B-C → B |
| **stripHtml()** | Removes all XML/HTML tags entirely. | Bold → Bold |
| **substringAfter("x")** | Returns everything after the first occurrence of x. | substringAfter("-") on A-B → B |
| **substringBefore("x")** | Returns everything before the first occurrence of x. | substringBefore("-") on A-B → A |
| **trim()** | Removes leading and trailing whitespace. | Test → Test |
| **upper()** | Converts the string to all uppercase. | urgent → URGENT |

## C: App Use Cases

This appendix provides practical scenarios for each App to demonstrate how inputs, property configurations, and resulting outputs correlate.

### 1. Create Data

**Scenario 1: Basic XML Generation with Variables**

- *Input (Private Data):** OrderID = 10045, Company = Acme Corp
- *Variables:** OrderID=[Job.PrivateData:Key="OrderID"], Company=[Job.PrivateData:Key="Company"]
- *Structure:** <Order><ID>{{OrderID}}</ID><Client>{{Company}}</Client></Order>
- *Type:** XML
- *Output File:** <Order><ID>10045</ID><Client>Acme Corp</Client></Order>

**Scenario 2: Dynamic JSON Payload Creation**

- *Input:** System date is March 27, 2026.
- *Structure:** {"status": "Approved", "date": "{{now()}}"}
- *Type:** JSON
- *Output File:** {"status": "Approved", "date": "2026-03-27"}

**Scenario 3: Plain Text / CSV Generation**

- *Input:** Headers = Name,Age,City | Row1 = John,30,Toronto
- *Structure:** {{H}}\n{{R}}
- *Type:** Other
- *Output File:** Name,Age,City / John,30,Toronto

### 2. Edit Data

**Scenario 1: Updating an Attribute**

- *Input File:** <Job priority="low"><File>art.pdf</File></Job>
- *Action 1:** Update | XPath: /Job/@priority | Type: Attribute Value | Value: high
- *Output File:** <Job priority="high"><File>art.pdf</File></Job>

**Scenario 2: Inserting a New Element**

- *Input File:** <Order><ID>99</ID></Order>
- *Action 1:** Insert | XPath: /Order | Type: Element | Name: Status | Value: Processed | Location: End
- *Output File:** <Order><ID>99</ID><Status>Processed</Status></Order>

**Scenario 3: Deleting a Node**

- *Input File:** <Record><Name>Sam</Name><InternalNotes>Secret</InternalNotes></Record>
- *Action 1:** Delete | XPath: /Record/InternalNotes
- *Output File:** <Record><Name>Sam</Name></Record>

## 3. Join Data

**Scenario 1: Merging Identical XML Structures**

- *Input:** file1.xml: <Item>A</Item> | file2.xml: <Item>B</Item>
- *XPath:** /* | Outgoing type: XML | Root element name: Items
- *Output File:** <Items><Item>A</Item><Item>B</Item></Items>

**Scenario 2: Joining JSON Objects into an Array**

- *Input:** 1.json: {"id": 1} | 2.json: {"id": 2}
- *Incoming type:** JSON | XPath: /* | Outgoing type: JSON | JSON type: Array
- *Output File:** [{"id": 1}, {"id": 2}]

**Scenario 3: Extracting and Merging Specific Nodes**

- *Input:** inv1.xml: <Invoice><Total>100</Total></Invoice> | inv2.xml: <Invoice><Total>250</Total></Invoice>
- *XPath:** /Invoice/Total | Outgoing type: XML | Root element name: Summary
- *Output File:** <Summary><Total>100</Total><Total>250</Total></Summary>

## 4. Select Data

**Scenario 1: Single Value Selection**

- *Input File:** <User><Email>test@example.com</Email></User>
- *Select 1 Name: **UserEmail |** XPath 1:** /User/Email/text()
- *Output (Private Data):** UserEmail = test@example.com

**Scenario 2: Multiple Values with Separator**

- *Input File:** <Tags><Tag>Red</Tag><Tag>Urgent</Tag></Tags>
- *Handler: **Separator |** Separator: **, |** Select 1 Name: **JobTags |** XPath 1:** /Tags/Tag/text()
- *Output (Private Data):** JobTags = Red, Urgent

**Scenario 3: Applying String Functions**

- *Input File:** <Order><Cost> 1,250.00 </Cost></Order>
- *Select 1 Name: **CleanCost |** XPath 1: **/Order/Cost/text() |** Function 1:** s.trim().mathNumber()
- *Output (Private Data):** CleanCost = 1250.00

## 5. Split Data

**Scenario 1: Standard XML Node Splitting**

- *Input File:** <Batch><Job>1</Job><Job>2</Job></Batch>
- *XPath: **/Batch/Job |** Keep context:** No
- *Output:** File 1: <Job>1</Job> | File 2: <Job>2</Job>

**Scenario 2: Splitting with Parent Context Retained**

- *Input File:** <Library><Book>A</Book><Book>B</Book></Library>
- *XPath: **/Library/Book |** Keep context:** Yes
- *Output:** File 1: <Library><Book>A</Book></Library> | File 2: <Library><Book>B</Book></Library>

**Scenario 3: Splitting JSON Arrays**

- *Input File:** [{"user": "Alice"}, {"user": "Bob"}]
- *Incoming type: **JSON |** XPath: ***/* |** Outgoing type:** JSON
- *Output:** File 1: {"user": "Alice"} | File 2: {"user": "Bob"}

## 6. Convert From XML

**Scenario 1: XML to JSON Conversion**

- *Input File:** <Person><Name>Jane</Name><Age>28</Age></Person>
- *Outgoing type:** JSON
- *Output File:** {"Person": {"Name": "Jane", "Age": "28"}}

**Scenario 2: XML to CSV**

- *Input File:** <Data><Row><ID>1</ID><Val>A</Val></Row><Row><ID>2</ID><Val>B</Val></Row></Data>
- *Outgoing type: **Spreadsheet |** XPath: **/Data/Row |** Type:** Comma Separated Values
- *Output File (.csv):** ID,Val / 1,A / 2,B

**Scenario 3: XML to Excel with Custom Headers**

- *Input File:** <Export><Item><sku>X1</sku><qty>5</qty></Item></Export>
- *Outgoing type: **Spreadsheet |** Type: **Excel 2007+ XML Format |** Custom headers:** sku=Part Number\nqty=Quantity
- *Output File (.xlsx):** Column A: "Part Number" (X1) | Column B: "Quantity" (5)

## 7. Convert To XML

**Scenario 1: JSON to XML**

- *Input File:** {"settings": {"theme": "dark", "version": 2}}
- *Incoming type:** JSON
- *Output File:** <JSON><settings _type="object"><theme _type="string">dark</theme><version _type="number">2</version></settings></JSON>

**Scenario 2: Spreadsheet to XML (Automatic Range)**

- *Input File (.xlsx):** Row 1 headers: Name, Role. Row 2: Admin, IT.
- *Incoming type: **Spreadsheet |** Range:** Automatic
- *Output File:** <Spreadsheet _type="object"><Row _type="object"><Name _type="string">Admin</Name><Role _type="string">IT</Role></Row></Spreadsheet>

**Scenario 3: Plain Text to XML**

- *Input File (.txt):** Hello World (Filename: input.txt)
- *Incoming type:** Text
- *Output File:** <TEXT filename="input">Hello World</TEXT>